W3docs

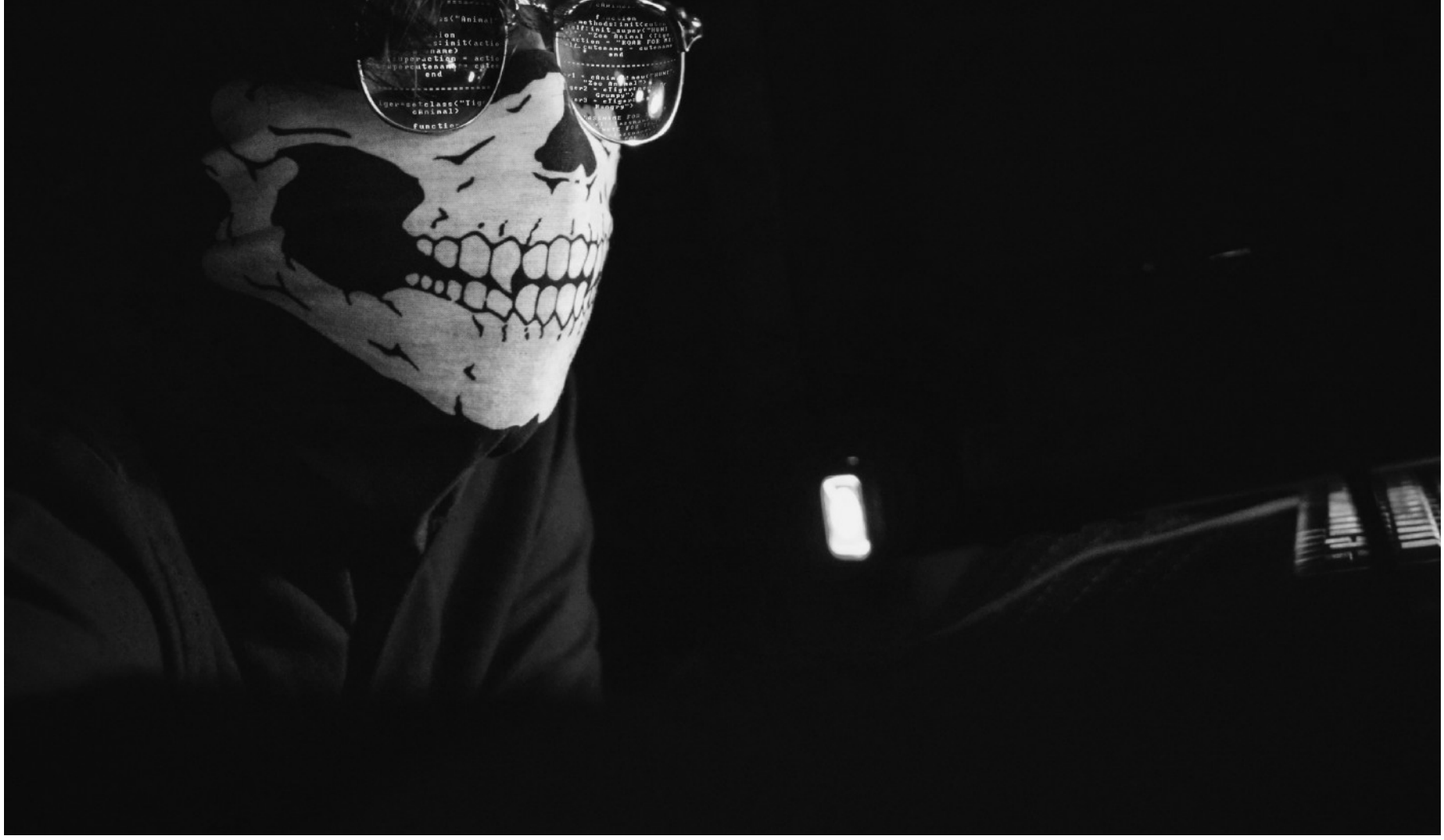


Photo by Nahel Abdul Hadi on Unsplash

An Ultimate Guide to SSH

W3docs Apr 16, 2020 · 3 min read

SSH keys appear to be commonly used worldwide. The keys are used for executing financial transactions, updating configurations, moving log data, file transfers, interactive logins by system administrators, and for other purposes.

What are SSH Keys?

SSH keys are a credential access that is used in SSH protocol (Secure Shell). Secure Shell is a network protocol that helps you to log in from one computer to another securely, as well as to manage networks,

operating systems, and configurations. Roughly said, SSH keys are nearly the same as passwords, but be sure that it is hundred times more secure to log into a server with SSH keys than use only “hard-to-decrypt” passwords because decrypting SSH is not an easy thing.

SSH keys come in pairs, and each of these pairs consists of a *public key* and a *private key*. These two are types of keys.

1. Public keys (or authorized keys) determine who can access the system.

2. Private (or identity keys) identify users and allow

their access.

When using SSH Key for Git means that you inform Git that your PC is authenticated for that specific GitHub account, and it won't ask you about access ever again because you have already given it your SSH Key.

How to Generate SSH Key?

The steps of creating an SSH Key for Linux and Mac is actually the same as these two operating systems provide modern terminal apps combining with the SSH package.

1. Open the terminal and run:

> *ssh-keygen -t rsa -b 4096 -C “your_email@example.com”*

2. Next, you should choose the file location:

> *> Enter a file in which to save the key (/Users/you/.ssh/id_rsa): [Press enter]*

Adding extra security won’t hurt you. So, here you need to add a passphrase to your SSH key.

> *> Enter passphrase (empty for no passphrase): [Type a passphrase]*
>
> *> Enter same passphrase again: [Type passphrase again]*

You can use SSH-agent to save your passphrase securely, so you don't have to re-enter it every time.

Adding New SSH Key to SSH-Agent

The responsibility of the SSH-agent tool is holding the private keys, as well as arranging requests to sign SSH requests with the private keys for security purposes.

1. Run the following code so as to make sure that the SSH-agent code is executing:

> *$ eval "$(ssh-agent -s)"*
>
> *> Agent pid 59566*

2. If it is running, continue the process by adding the new SSH key to the local SSH agent:

```
ssh-add -K /Users/you/.ssh/id_rsa
```

How To Add SSH Key To GitHub Account

1. Your first step should be copying the SSH Key to your clipboard.

2. If the filename differs from the one of the code, match the filename to the current setup:

```
$ sudo apt-get install xclip
```

 # Downloads and installs xclip. If you don't have 'apt-get', you should use another installer (e.g. 'yum')

$ xclip -sel clip < ~/.ssh/id_rsa.pub

 # Copies the content of the id_rsa.pub file to the clipboard

3. The next step is clicking your profile photo in the top right corner of any page and then click Settings.

4. Go to the SSH and GPG keys in the user settings sidebar.

5. Next, click New SSH Key.

6. Type Title and add a description to the SSH Key.

7. Paste your key into the “Key” field.

8. And, finally, hit “Add SSH Key”.

Congratulations, you have successfully added your PC’s SSH key to your GitHub account.

. . .

The crucial part in public-key authentication is that it allows one server to access another server without typing

the password again. This is the reason that is why it gains popularity and is so commonly used for file transfers. The single sign-on operation provides convenience for the users, thus, making it a popular feature.

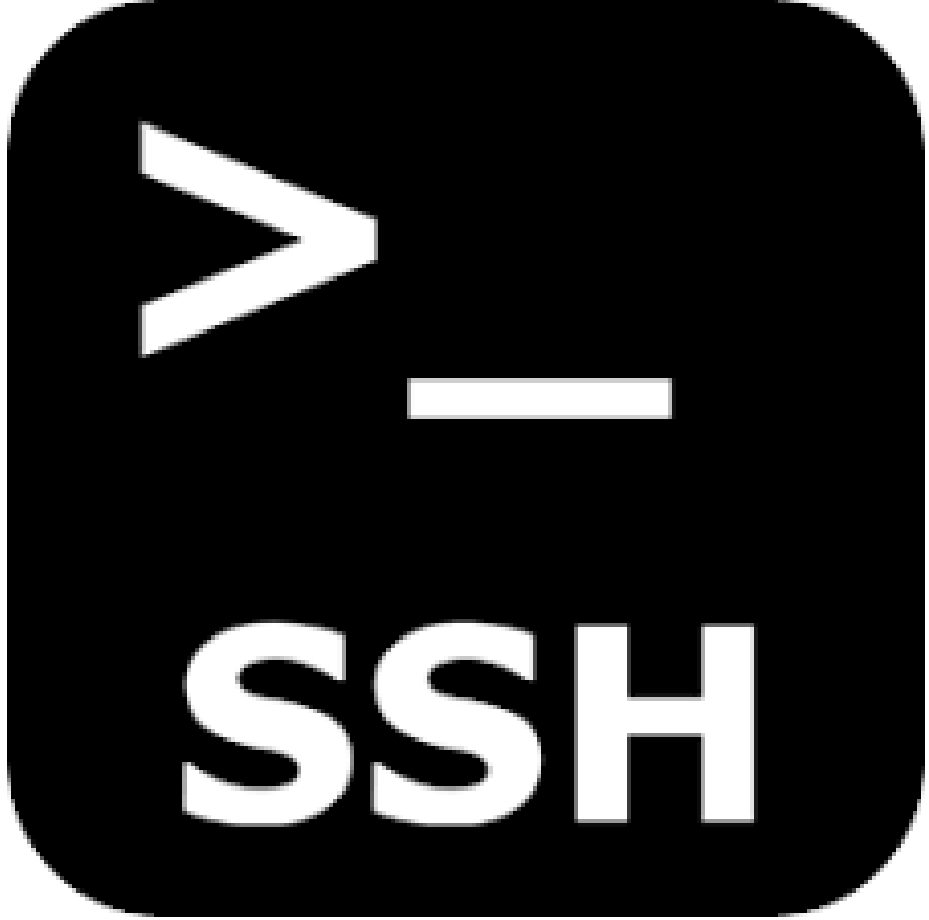
Check out more git commands and snippets, and get into the Git world to explore new ideas!

Git Github Productivity

Mark

What is SSH key?

Mark Jul 5, 2018 · 2 min read



SSH key is a credential in SSH protocol. The function is similar to your username and password. SSH key is a key pair, included a privacy key and a public key.

Public key can be shared to others and can be easily generated from privacy key. Privacy key **MUST NOT** be shared to others and cannot be easily generated from public key (based on current computational power).

User Keys & Host Keys

SSH key implementation supports host to access remote server and remote server access host (you can think as your personal computer).

When user accesses remote server, it is called **User Keys**.

When remote server access host (e.g. your PC), it is called **Host Keys**.

How User Keys work?

We use an example to explain how you use your computer to access a remote server.

1. client generates public key and privacy key [`ssh-keygen`]
2. client copies public key to remote server [`ssh-copy-id`]
3. server stores the public key
4. when client wants to connect to remote server, it initiates a connection to server via SSH protocol [`ssh`

`<username>@<ipAddress>`]

5. remote server receives the connection from client
6. remote server identifies which public key should be used based on protocol
7. remote server uses public key to encrypt a random message
8. remote server sends the encrypted message to client
9. client uses private key to decrypt the message
10. client sends the decrypted message and previous

session ID to remote server

11. remote server verifies the decrypted message from client, which is matched the sent message or not

12. if match, client gains access to remote server

How Host Keys work?

The implementation is similar to user keys. The public key and privacy key is stored in remote server and the public key is stored in host.

Why I need to use SSH key?

It can protect your data via the Internet, e.g. prevent

man-in-the-middle.

Encryption and Size of SSH key

There are different size of the SSH keys, one of the suggested encryption is `RSA 2048-bit encryption`. The key is a 617-digit number.

Where can I find my key?

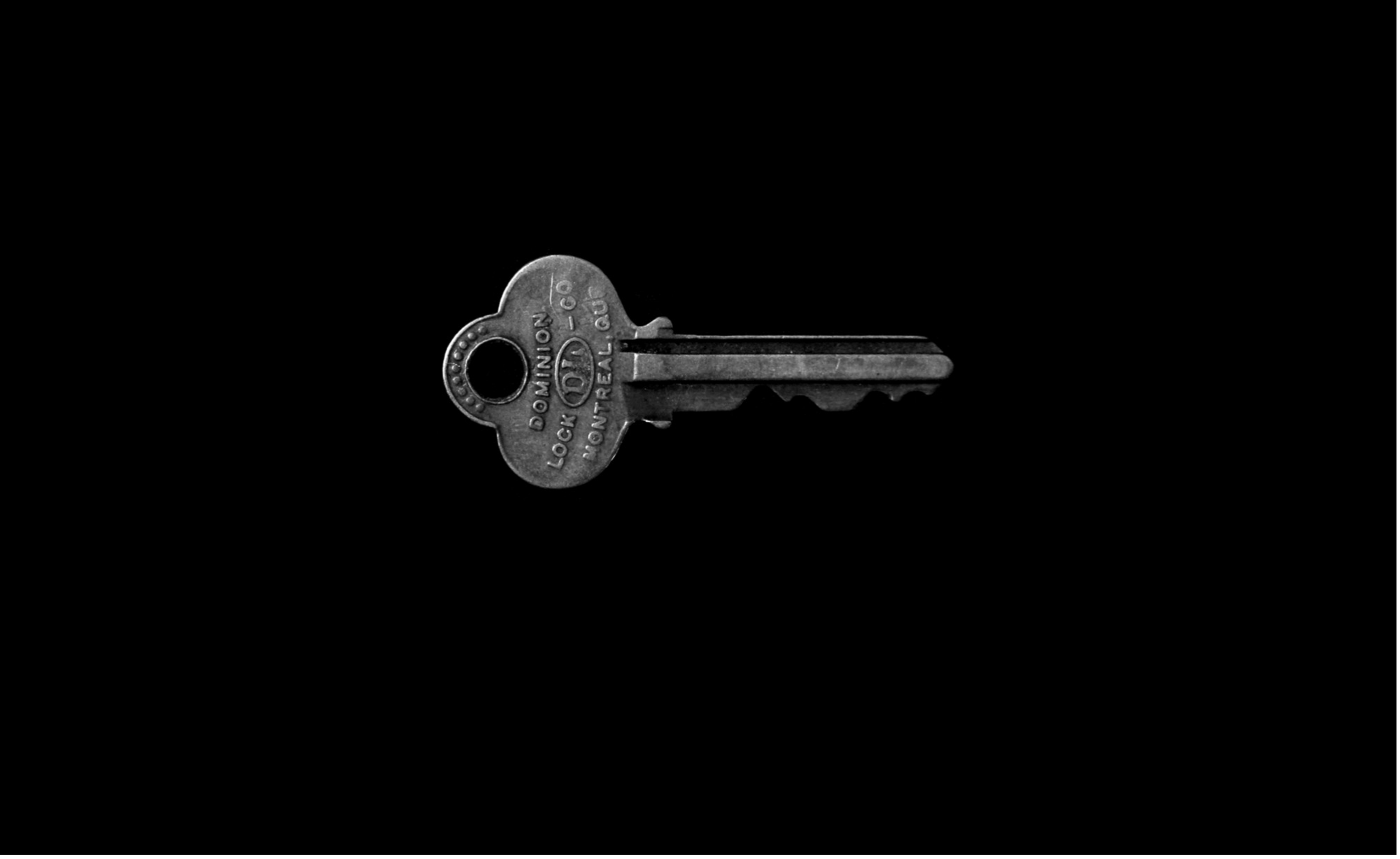
In your computer, it usually store in `.ssh/`. In remote server, it suggests to use a management key tools to manage the key due to the number of key. It can be a large number if your services is used by many parties.

Ssh | Explanation | Ssh Keys | User Keys | Host Keys

Upgrade Your SSH Key to Ed25519

Risan Bagja Follow

Jan 9, 2018 · 4 min read



Upgrade your SSH key! Photo by Matt Artz on Unsplash

If you're a DevOps engineer or a web developer, there's a good chance that you're already familiar and using the SSH key authentication on a daily basis. Whether it's for logging into the remote server or when pushing your commit to the remote repository. It provides us with better security than the traditional password-based authentication.

But, when is the last time you created or upgraded your SSH key? And did you use the latest recommended public-key algorithm? If it was more than five years ago and you generated your SSH key with the default options, you probably ended up using RSA algorithm with key-size less than 2048 bits long.

Check Available SSH Keys on Your Computer

To check all available SSH keys on your computer, run the following command on your terminal:

```
for key in ~/.ssh/id_*; do ssh-keygen -l -f
"${key}"; done | uniq
```

Your SSH keys might use one of the following algorithms:

- **DSA**: It's unsafe and even no longer supported since OpenSSH version 7, you need to upgrade it!
- ⚠ **RSA**: It depends on key size. If it has 3072 or 4096-bit length, then you're good. Less than that, you probably want

to upgrade it. The 1024-bit length is even considered unsafe.

- **ECDSA**: It depends on how well your machine can generate a random number that will be used to create a signature. There's also a trustworthiness concern on the NIST curves that being used by ECDSA.

- ✅ **Ed25519**: It's the most recommended public-key algorithm available today!

Some Ed25519 Benefits

The Ed25519 was introduced on OpenSSH version 6.5. It's the EdDSA implementation using the Twisted Edwards curve. It's

using elliptic curve cryptography that offers a better security with faster performance compared to DSA or ECDSA.

Today, the RSA is the most widely used public-key algorithm for SSH key. But compared to Ed25519, it's slower and even considered not safe if it's generated with the key smaller than 2048-bit length.

The Ed25519 public-key is compact. It only contains 68 characters, compared to RSA 3072 that has 544 characters. Generating the key is also almost as fast as the signing process. It's also fast to perform batch signature verification with Ed25519. It's built to be collision resilence. Hash-function

collision won't break the system.

Are you ready to switch to Ed25519?

Generating Ed25519 Key

You can have multiple SSH keys on your machine. So you can keep your old SSH keys and generate a new one that uses Ed25519. This way you can still log in to any of your remote servers. Then slowly replace the authorized key on your remote servers one by one with the newly generated Ed25519 public-key.

Open up your terminal and type the following command to generate a new SSH key that uses Ed25519 algorithm:

Generate SSH key with Ed25519 key type

You'll be asked to enter a passphrase for this key, use the strong one. You can also use the same passphrase like any of your old SSH keys.

- `-o` : Save the private-key using the new OpenSSH format rather than the PEM format. Actually, this option is implied when you specify the key type as `ed25519`.

- `-a` : It's the numbers of KDF (Key Derivation Function) rounds. Higher numbers result in slower passphrase verification, increasing the resistance to brute-force password cracking should the private-key be stolen.

- `-t` : Specifies the type of key to create, in our case the Ed25519.

- `-f` : Specify the filename of the generated key file. If you want it to be discovered automatically by the SSH agent, it must be stored in the default `.ssh` directory within your home directory.

- `-C` : An option to specify a comment. It's purely

informational and can be anything. But it's usually filled with `<login>@<hostname>` who generated the key.

Adding Your Key to SSH Agent

You can find your newly generated private key at `~/.ssh/id_ed25519` and your public key at `~/.ssh/id_ed25519.pub`. Always remember that your public key is the one that you copy to the target host for authentication.

Before adding your new private key to the SSH agent, make sure that the SSH agent is running by executing the following command:

```
eval "$(ssh-agent -s)"
```

Then run the following command to add your newly generated Ed25519 key to SSH agent:

```
ssh-add ~/.ssh/id_ed25519
```

Or if you want to add all of the available keys under the default `.ssh` directory, simply run:

```
ssh-add
```

Notes for macOS User

If you're using macOS Sierra 10.12.2 or later, to load the keys automatically and store the passphrases in the Keychain, you need to configure your `~/.ssh/config` file:

```
Host *
  AddKeysToAgent yes
  UseKeychain yes
  IdentityFile ~/.ssh/id_ed25519
  IdentityFile ~/.ssh/id_rsa # Keep any old key
files if you want
```

Once the SSH config file is updated, add the private-key to the SSH agent:

```
ssh-add -K ~/.ssh/id_ed25519
```

Specifying Specific Key to SSH into a Remote Server

The SSH protocol already allows the client to offer multiple keys on which the server will pick the one it needs for authentication. However, we can also specify a specific private-key to use like so:

```
ssh -i ~/.ssh/id_ed25519 john@198.222.111.33
```

Or you can even add an entry to the `~/.ssh/config` file to configure these options:

```
Host awesome
  HostName 198.222.111.33
  User john
  IdentityFile ~/.ssh/id_ed25519
  IdentitiesOnly yes
```

Once it's saved, later you can SSH to your target host like this:

```
ssh awesome
```

Awesome right? 😎

Originally published at risanb.com on 29 November 2017.

Ssh Servers DevOps Security Ed25519

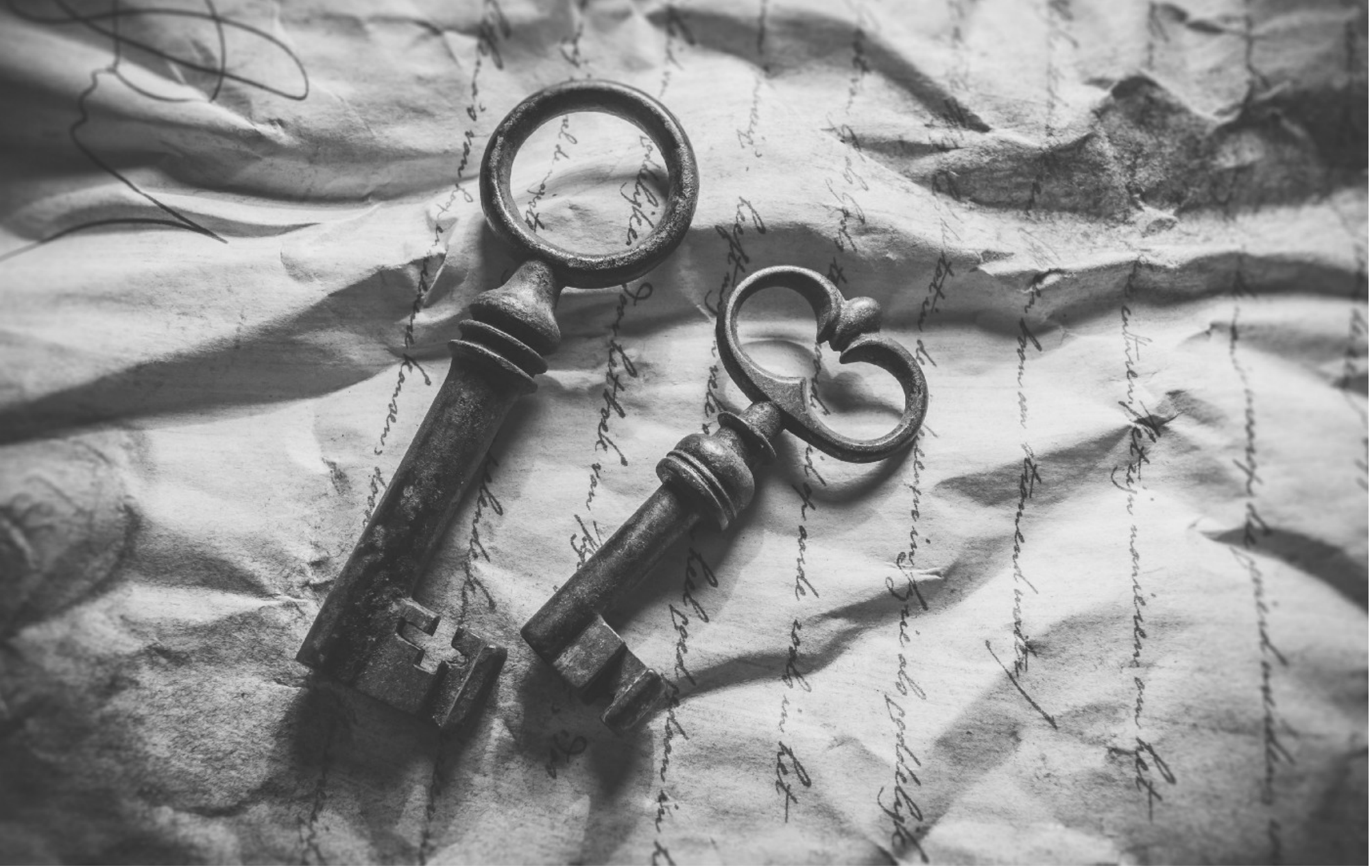
How to setup two two different SSH-Keys for one GIT Host

A small HowTo Guid on how to use different SSH keys for different users on the same GIT Host like GitHub or Bitbucket.

Thomas Strohmeier

Follow

Oct 14 · 1 min read



As a development agency we are regularly working with

different customers and different VCS platforms. Sometime it is required to create a dedicated account for a project with a new user. Simply adding a new SSH-Key leads to the problem of using the wrong SSH-key to the account. Here I want to provide you a simple solution for this problem:

1. Creating a SSH Config File

First we need a SSH config file. This is usually located at `~/.ssh/config`.

```
  #default account
  Host github.org
          HostName github.com
```

```
        User git
        IdentityFile ~/.ssh/id_rsa

#account b account account
Host github.com-accb
        HostName github.com
        User git
        IdentityFile ~/.ssh/account_b_id_rsa
```

2. Add SSH-Key of Account B

```
$ ssh-add ~/.ssh/account_b_id_rsa
```

3. Using the Alias Domain in git clone

If you now want to use the Account B in git clone simply replace the host with the alias:

```
$ git clone git@github.com-
accb:companyB/repo.git
```

That's it. ✅

PS: If you have an idea how to improve this setup, I would be happy If you leave a comment 😉

Git Ssh Ssh Keys Bitbucket Github